

## Classes e Struts 01 - Intro

Classes e Struts são estruturas flexíveis que constituem o código do nosso programa, você pode definir propriedades e métodos que adicionam funcionalidades as suas classes e estruturas.

No **Swift** não há necessidade de criar arquivos de interface e implementação separados (como em C++), classes e struts são definidas em um único arquivo (semelhante a Java).

### O que classes e estrutura possuem em comum?

- Definem propriedades
- Definem métodos
- Definem construtores
- Podem ser estendidos para expandir funcionalidades
- Podem assinar protocolos (protocolos exigem a existência de métodos)

### Qual o Diferencial de Classes em relação às estruturas?

- Herança
- Deinitializers permitem que instancias liberem recursos
- Type Casting permite que você avalie e interprete o tipo de uma instância de classe em tempo de execução
- Mais de uma referência para a instância de uma classe

- **IMPORTANTE** Struts são **Value Types** e são passados por cópia, enquanto classes são **Reference Types** e são passados por referência, o que significa que struts não usam o contador de referência. Veja o Exemplo abaixo.

==> Você pode criar uma estrutura (struct) PontoR2 com atributos x e y inteiros?

==> Você pode criar uma classe (class) quadrado com um ponto superior a esquerda (topLeft) do tipo PontoR2 e mais um atributo para descrever o lado do quadrado?

```
1  
2 struct Resolution {  
3     var width = 0  
4     var height = 0  
5 }  
6  
7 class VideoMode {  
8     var resolution = Resolution()  
9     var frameRate = 0.0  
10    var name: String?  
11 }  
12  
13
```

Run

## Refêrencia vs Valor

Em **Swift** Classes e Closures são Reference Types (tipos que trabalham com referência), todo o resto é Value Type (tipos que trabalham como valores), inclusive as struts são Value Type. Struts possuem inicializadores autogerados com todas as propriedades ou seja um construtor que recebe todos os atributos.

==> Aqui temos uma prova que struts são value types. Você pode criar uma prova para arrays:

```
1 struct PontoR2 {
2     var x : Int = 0
3     var y : Int = 0
4 }
5
6
7 print("Inicialização de p1 e p2")
8 var p1 = PontoR2(x:10, y:10) // Inicializador autogerado para iniciar x e y
9 print("p1.x=\(p1.x)")
10
11 var p2 = PontoR2(x:20, y:20)
12 print("p2.x=\(p2.x)")
13
14 p1 = p2 //aqui é feita uma cópia de valores p1 para p2
15 print("Depois da Cópia")
16 print("p1.x=\(p1.x)")
17 print("p2.x=\(p2.x)")
18
19 p2.x=30 //Veja que isso não interfere em p1
20 print("Depois da atribuição a p2")
21 print("p1.x=\(p1.x)")
22 print("p2.x=\(p2.x)")
23
24
```

Run

- **IMPORTANTE** Assim fica claro o que é um Value Type. Ou seja quando ocorre atribuições em um elemento da estrutura (p1=p2) nada acontece com o elemento que recebeu a cópia (nesse caso alterações em p2 não interferem em p1).

Veja um exemplo de como funciona um Reference Type.

==> Você pode fazer um exemplo usando sua classe quadrado?

```
1 class Flor {
2     var cor : String = "brnaca"
3
4     init (cor : String) {
5         self.cor = cor
6     }
7
8     func show () {
9         print (self.cor)
10    }
11
12 }
13
14 print ("Inicialização")
15 var lirio = Flor (cor: "banca")
16 var azaleia = Flor (cor: "rosa")
17 lirio.show()
18 azaleia.show()
19
20 print ("Lírio agora aponta para azaleia (ref para azaleia)")
21 lirio = azaleia
22 lirio.show()
23 azaleia.show()
24
25 print ("Modificação e azaleia afeta lírio")
26 azaleia.cor = "violeta"
27 lirio.show()
28 azaleia.show()
29
30
31
```

Run

- **IMPORTANTE** Assim fica claro o que é um Reference Type. Ou seja quando ocorre atribuições em um objeto de uma classe (lirio = azaleia) esse objeto (lirio) recebe uma referência para o objeto (azaleia) e não uma cópia. Assim, alterações em azaleia irão refletir em lírio.

BETA

==> voce pode fazer modificações no exemplo de modo a incluir comentários do que esta acontecendo:

```
1
2 struct Resolution {
3     var width = 0
4     var height = 0
5 }
6
7 class VideoMode {
8     var resolution = Resolution()
9     var frameRate = 0.0
10    var name: String?
11 }
12
13 let hd = Resolution(width: 1920, height: 1080)
14 var cinema = hd
15 cinema.width = 2048
16 print(hd.width)
17
18 let videoMode = VideoMode()
19 let anotherMode = videoMode
20 anotherMode.frameRate = 24.0
21 print(videoMode.frameRate)
22
```

Run

###Quando usar structs:

- Quando queremos encapsular dados relativamente simples
- Quando Propriedades do modelo também são **Value Types**
- Quando o modelo não precisa herdar nenhuma propriedade ou comportamentos de modelos existentes ( Pode ser alcançado através de protocolos também )
- Quando os dados encapsulados devem ser copiados e não referenciados.


==> Veja o exemplo e crie o seu próprio. Faça um comentário do porquê justificando a escolha da struct e não class para seu exemplo.

```
1
2 struct PointR3 {
3     var x: Double
4     var y: Double
5     var z: Double
6 }
7
8
```

Run

## Suggested playgrounds

### Classes and Struts - Swift (parte...

By  1,074 0 0

### Classes and Struct - (Properties)

By  1,247 1 0

### TALK 01 - seja bem vindo!

By  812 1 0

### Swift - Loops

By  1,386 0 0

BETA

Open Source Your Knowledge: become a Contributor and help others learn.


[Create New Content](#)

0 Comments

 Login ▼

Start the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name

Email

Password

- ☐ I agree to Disqus' [Terms of Service](#)
- ☐ I agree to Disqus' processing of email and IP address, and the use of cookies, to facilitate my authentication and posting of comments, explained further in the [Privacy Policy](#)
- ☐ I agree to additional processing of my information, including first and third party cookies, for personalized content and advertising as outlined in our [Data Sharing Policy](#)
- ☐ I'd rather post as a guest

